

Chapter 30: Handling Errors and Messages

Overview

When a user enters screen input, the transaction must check the input's validity before using it. The SAP System provides error-handling features that simplify field-checking as much as possible. These features include keywords for programming error-handling, as well as aspects of the dialog-processing runtime environment:

- automatic field checks (performed by the system)
Some field checks are performed automatically by the system, based on information stored in the ABAP/4 Dictionary.
- the FIELD and CHAIN statements (in the flow logic language)
The FIELD and CHAIN flow logic statements let you program your own field checks. FIELD and CHAIN tell the system which fields you are checking, and whether the system should perform checks in the flow logic or call an ABAP/4 module. If errors are found, the system enters an error dialog with the user.
- the MESSAGE statement (in ABAP/4)
The MESSAGE statement (in ABAP/4) lets you put out messages from an ABAP/4 program. An ABAP/4 program notifies the system of errors by putting out an error message or warning. In response, the system enters its error dialog with the user.
- an error dialog (performed by the system)
Errors can be detected either by the system or by an ABAP/4 module. In either case, when an error is found, the system automatically re-displays the screen and puts out a message.

Errors are most often field-specific. In the re-display, the field (or fields) causing the error is input-enabled, and all other fields are disabled. The system places the cursor in the error field, and requires the user to re-enter the input. The field checks are then repeated.

The following topics provide information:

Introduction to Error Processing

Checking Screen Fields for Validity

Issuing Messages

Example Transaction: Checking Field Input

Contents

Introduction to Error Processing	30-3
Checking Screen Fields for Validity.....	30-6
Understanding Automatic Field Checks.....	30-7
Checking Fields in the Screen Flow Logic	30-8

Checking Fields in ABAP/4	30-9
Checking a Single Field	30-9
Checking Multiple Fields	30-10
Making Module Calls Conditionally	30-11
Conditional FIELD Statements	30-12
Conditional CHAIN Statements	30-12
Avoiding Automatic Field Checks	30-13
Issuing Messages	30-14
Sending a Message.....	30-15
Creating a Message Class.....	30-18
Creating Messages	30-19

Introduction to Error Processing

In normal dialog processing, the transaction progresses from one screen to the next. However, if an error occurs, the system redisplay the screen where the error occurred. A message is displayed, and if the error involved field input, the field is input-enabled. (All other fields retain fixed values.) How does this look to the user, and how does your program tell the ABAP/4 processor that a re-display is necessary?

Look at transaction TZ31 for an example of error-handling. TZ31 (development class SDWA) is a small transaction for displaying and updating flight information. The transaction lets the system perform automatic field checking, but also contains its own logic to conduct additional error-checks.

Normally, when you use the transaction, you enter airline and flight identifiers and press **ENTER**. The system then displays all field details in update-mode. To make your changes, you type in the new information and save.

What happens when your input is wrong? Suppose you just pressed **ENTER** without typing in any of the required information. (Fields with “?” in them are required-input fields.) The system checks for this automatically, and sends you a message:

Change Flight Data

Flight data Edit Goto System Help

Other flight

Airline carrier ?
Flight number ?

Flight data

From city
Dep. airport
Destination
Dest. airport
Flight time 00:00:00
Departure 00:00:00
Arrival 00:00:00
Distance
Dist. in

Flight type

☒ Sched. flight
☐ Charter flight

E: Required entry not made

TZ31 also checks for things the system neglects. For instance, what if when updating the display, you enter an airport code that doesn't exist? The program sends you a message:

Other flight

Airline carrier ab
Flight number 17

Flight data



In this screen, only the airport field can be changed. All other input is fixed. When you then correct the airport and re-enter, the transaction continues with its other processing.

By experimenting with TZ31, you will see that several kinds of field checks take place. Some are handled automatically by the system, and some by the program:

- Do the fields that require input have input in them? (**automatic**)
The *Airline* and *Flight-number* fields have the required-input attribute in the Screen Painter. The system automatically checks that these fields get input from the user.
- Does the airline entered exist? (**automatic**)
In the Screen Painter, the *Airline* field is declared as the table field SPFLI-CARRID. In the Dictionary, the CARRID field has a foreign key relationship with check table SCARR. As a result, the system automatically checks that all input to SPFLI-CARRID is contained in SCARR.

Checking Screen Fields for Validity

- Does the flight-number exist for this airline? (**ABAP/4**)
An ABAP/4 module (CHECK_FLIGHT) in transaction TZ31 checks that the flight number entered exists for the given airline.
- Departure/arrival cities: do they exist? (**automatic**)
The *Departure city* field (SPFLI-CITYFROM) is a foreign key, with check table SGEOCITY. The system automatically checks that the input to this field is found in the SGEOCITY table.
- Departure/arrival airports: do they exist? (**ABAP/4**)
The ABAP/4 modules check that the airports entered exist.

The rest of this chapter tells you how to program error-handling:

Checking Screen Fields for Validity

Issuing Messages

For a discussion of how transaction TZ31 is implemented, see *Example Transaction: Checking Field Input*.



Note

If you want to play with transaction TZ31 in the system, remember that you can use the Data Browser to find out valid flight numbers for a given airline code. To access the Data Browser, get into the Object Browser (in the Workbench) and select *Environment* → *Data Browser* → *Table contents*.

Checking Screen Fields for Validity

The R/3 System offers various methods for checking screen fields:

- Automatic field checks, performed by the system
The automatic field checks are the easiest to use, when they are suitable. The system makes these checks based on field information you can store in the ABAP/4 Dictionary.
- Checks performed in the screen flow logic
You can specify some field checks in the flow logic for a screen. You do this with the flow logic statement FIELD...VALUES..., which specifies a list of possible values for a screen field. The system checks screen field input against these values without even entering an ABAP/4 module. As a result, you can design and test a screen without having any ABAP/4 code available.
- Checks performed in ABAP/4
When the automatic field checks and the FIELD...VALUES... flow logic statement are not flexible enough, you can code ABAP/4 modules to perform field-checking. To trigger a call to the module, you code a FIELD...MODULE... flow logic statement in your screen flow logic.

See the following topics for more explanation:

Understanding Automatic Field Checks

Checking Fields in the Screen Flow Logic

Checking Fields in ABAP/4

Understanding Automatic Field Checks

The system automatically carries out certain validity checks on screen fields. These checks are performed after the user enters input, and before any PAI processing has begun. The types of automatic checks performed are:

- Required input

In the Screen Painter, you can set a field's required-input (*Req.entry*) attribute. When you do, the system requires the user to enter input into the field before entering PAI processing.

- Proper data format

In the Screen Painter, each field has a data format. This format limits the kinds of input that can be valid. For example, a DATS field (a date field) is an 8-character string in YYYYMMDD format. All characters must be numbers. The substrings MM and DD must be less than or equal to 12 and 31 respectively. For the given month value entered, the system also checks that the day value is valid.

When the user enters input that does not match the data format requirements, the system re-displays the screen until the input is corrected.

- Valid value for the field

In the Dictionary, there are two ways to restrict field values to an allowable set. The field can have a **foreign key relationship** with another table, or its domain can specify a **fixed-value list** for the field. For foreign key fields, the system checks that the user's input value can be found in the related check table. For fields with defined fixed-value lists, the system ensures that the user's input value is one of the values in the list.

For foreign-key fields, you can store a predefined error message with the field.

If any field fails a validity test, the system re-displays the screen and requires the user to change his input. Once re-entered, all automatic checks are repeated for fields in which the user enters new input.

To extend the automatic field checks, you can define field checks of your own. For how to do this, see:

Checking a Single Field

Checking Multiple Fields

**Note**

Sometimes you want to code processing that should happen *before* the system carries out any automatic checks. This processing usually handles exit requests. (For example, when the user wants to return from a transaction, there is no reason to make him enter input in required fields.) To code logic that should execute before the system performs its automatic field checks, use the flow logic command **MODULE ... AT EXIT-COMMAND**. For information, see *Avoiding Automatic Field Checks*.

Checking Fields in the Screen Flow Logic

Use the **FIELD...VALUES** flow logic statement to check field values in screen flow logic. With this statement, you specify a list of possible values, and the system compares the field input against this list. Syntax for the statement is:

```
FIELD <screen field>  VALUES (<value-list>)
```

The <value-list> can include individual values or value intervals. Surround all values with single quotes. For example:

```
FIELD SBOOK-CARRID VALUES ('AA', 'LH', 'US').  
FIELD SBOOK-CONNID VALUES (BETWEEN '1' AND '10', NOT '3').
```

If the value entered by the user is not equal to one of the values in the list (or fit within a specified interval), the system redisplay the screen for new input.

The following table shows all possible formats for the <value-list>.

<i>Comparison</i>	<i>Syntax</i>
Single value	('<value>')
Complement of single values	(NOT '<value>')
Several single values, or complements	('<value1>', '<value2>', NOT '<value3>')
Value interval	(BETWEEN '<value>' AND '<value>')
Everything outside a certain interval	(NOT BETWEEN '<value>' AND '<value>')

**Note**

To use the **FIELD...VALUES** option, the fields for comparison must have data type **CHAR** or **NUMC**. Also remember that all values given in single quotes should be capitalized.

Checking Fields in ABAP/4

To program field checks in ABAP/4, you use the FIELD and CHAIN flow logic language statements. The following form of the FIELD statement lets you call ABAP/4 modules that make field checks:

```
FIELD <field> MODULE <module>.
```

The FIELD statement may contain more than one MODULE call:

```
FIELD <field>:  MODULE <module1>,  
              MODULE <module2>.
```

You can also specify more than one field in a FIELD statement. This is especially useful when you want to chain field-checks together with the CHAIN statement. For example, both forms of the FIELD statement below are allowed:

```
CHAIN.  
  FIELD <field1>.  
  FIELD <field2>.  
  FIELD: <field3>, <field4>, ... <fieldn>.  
  MODULE <module1>.  
  MODULE <module2>.  
ENDCHAIN.
```

When an ABAP/4 module finds an error, it puts out an error message or warning to notify the user. Issuing these messages alerts the system that an error dialog is needed. The system redisplay the screen, requiring the user to enter a new value for the erroneous field. All other fields are input-disabled. With the CHAIN statement, if an error is found, the screen is re-displayed with all fields in the chain input-enabled.

The following topics provide information:

Checking a Single Field

Checking Multiple Fields

Making Module Calls Conditionally

Avoiding Automatic Field Checks

Checking a Single Field

The FIELD...MODULE flow logic statement lets you tie validity-checking to particular screen fields. FIELD...MODULE triggers calls to ABAP/4 modules that check particular fields and send error messages if errors are found. In this case, the system re-displays the screen, disabling input for all fields but the ones specified.

To notify the system that an error has been found, the ABAP/4 module must put out either an error message (type E) or a warning (type W). For example:

```
**** Screen flow logic: ****
```

```
PROCESS AFTER INPUT.  
  FIELD SPFLI-AIRPFROM  
    MODULE CHECK_FR_AIRPORT.
```

```
**** ABAP/4 module: ****
```

Checking Screen Fields for Validity

```
MODULE CHECK_FR_AIRPORT INPUT.  
  SELECT SINGLE * FROM SAIRPORT  
    WHERE ID = SPFLI-AIRPFROM.  
  IF SY-SUBRC NE 0. MESSAGE E003 WITH SPFLI-AIRPFROM. ENDIF.  
ENDMODULE
```

The error message causes the system to redisplay the screen and require new input from the user. In the re-display, only field SPFLI-AIRPFROM can take new input. All other fields are input-disabled.

For more information about error messages, see *Issuing Messages*.

How the FIELD statement transfers data

The system transfers data from the screen to the ABAP/4 program only once per screen display. Normally this happens at the beginning of PAI processing, after the system has performed its automatic field checking.

However there are exceptions to this. Transfer of data for fields mentioned in a FIELD statement is delayed until execution actually reaches the FIELD statement. If the screen field occurs in more than one FIELD statement, its value is transferred only at the first FIELD statement in which the field occurs.

The time of transfer is important because you should not use a field in an ABAP/4 module before it has been transferred from the screen. If you do, the ABAP/4 field will contain the value it had before the user entered screen input. This value can be either the value left over from the last screen, or even an invalid value.

Using a field in multiple FIELD statements

Sometimes you need to specify the same field in multiple FIELD statements. This makes error processing slightly more complex. When a module finds an error, the system re-displays the screen and restarts PAI processing with the corrected input. However, the system cannot simply restart with the FIELD statement containing the module. The field in error may also have occurred in some earlier FIELD or CHAIN statement. The system has a special procedure for determining where to restart processing in this case. For more information, see *Restarting PAI after an error dialog*.

Checking Multiple Fields

Sometimes you want to check several fields as a group. To do this, include the fields in a FIELD statement, and enclose everything in a CHAIN-ENDCHAIN block. A CHAIN statement is used in example transaction TZ31:

****** Screen flow logic: ******

```
CHAIN.  
  FIELD: SPFLI-CARRID, SPFLI-CONNID.  
  MODULE CHECK_FLIGHT.  
ENDCHAIN.
```

****** ABAP/4 module: ******

```

MODULE CHECK_FLIGHT INPUT.
  SELECT SINGLE * FROM SPFLI
    WHERE CARRID = SPFLI-CARRID
    AND CONNID = SPFLI-CONNID.
  IF SY-SUBRC NE 0.
    MESSAGE E005 WITH SPFLI-CARRID SPFLI-CONNID.
  ENDIF.
* .....
ENDMODULE.

```

In a chain block, all fields are mutually dependent. When an error is found inside a chain, the screen is re-displayed, and all fields found anywhere in the chain are input-enabled. All non-chain fields remain disabled. After the user re-enters his input (to one of the chain fields), PAI is restarted and all statements in the chain are re-executed as a unit.

Chains can include any other permissible flow logic-language statements. Also, chains can include more than one FIELD statement. In general, all FIELD statements should occur at the beginning of the CHAIN block.

```

CHAIN.
  FIELD: A, B, C.
  FIELD: D, E, F.
  MODULE X.
  MODULE Y.
ENDCHAIN.

```



Note

It is allowed, but not really sensible, to add a MODULE statement onto a FIELD statement contained in a CHAIN block:

```

CHAIN.
  FIELD F1.
  FIELD: F2, F3  MODULE m1.   "(No period after F3)
  MODULE m2.
ENDCHAIN.

```

If module *m* finds an error, it opens *all* chain fields for input in the re-display, not just F2 and F3. Using FIELD...MODULE in this way is only sensible when you are using one of the AT- or ON-conditions. For more information, see *Conditional CHAIN Statements*.

Making Module Calls Conditionally

You can place conditions on the module calls you make from a screen's flow logic. For example, you can specify that a module should only be called if a given field has a non-initial value in it:

```
FIELD X MODULE CHECK_FIELDX ON INPUT.
```

With the conditional forms of the FIELD statement, you can prevent unnecessary module calls. Particularly when you are updating table entries, conditional calls can improve performance substantially. The following topics provide information:

Conditional FIELD Statements

Conditional CHAIN Statements

Conditional FIELD Statements

The FIELD...MODULE flow logic statement becomes conditional when you add ON- and AT-conditions. Use the following conditions to specify when the module should be called:

- **ON INPUT**

The ABAP/4 module is called only if the field contains a value other than its initial value. This initial value is determined by the field's data type: blanks for character fields, zeroes for numerics. If the user changes a field value back to its initial value, ON INPUT does not trigger a call. (Contrast this with the ON REQUEST call, which does trigger the in this case.)

- **ON REQUEST**

The ABAP/4 module is called only if the user has entered a value in the field value since the last screen display. The value counts as changed even if the user simply types in the value that was already there.

In general, the ON REQUEST condition is triggered through any form of "manual input". The system considers the following ways of setting a field as manual input:

- Actual user input
- SET PARAMETER field input (both manual and automatic)
- HOLD DATA field input
- Parameter input for a parameter transaction (CALL TRANSACTION...USING)
- Global fields used to customize your system (these specify automatic settings for fields for certain fields)

Any of these fulfill the ON REQUEST condition and would trigger the module call.

- **ON *-INPUT**

The ABAP/4 module is called if the user has entered a "*" in the first character of the field, and the field has the attribute **-entry* in the Screen Painter. You can use this option in exceptional cases where you want to check only fields with certain kinds of input.



Note

Some of these conditions apply to FIELD statements alone, and others to FIELD statements inside CHAIN blocks. In particular, the ON- and AT-conditions have a special meaning in FIELD statements that contain multiple fields, but are not enclosed in a CHAIN block. See *Checking Multiple Fields* for details.

Conditional CHAIN Statements

To make the module calls in a CHAIN conditional, there are two options:

- **ON CHAIN-INPUT**

Similar to ON INPUT. The ABAP/4 module is called if *any one* of the fields in the chain contains a value other than its initial value (blanks or nulls).

- **ON CHAIN-REQUEST**

This condition functions just like ON REQUEST, but the ABAP/4 module is called if *any one* of the fields in the chain changes value.

For example:

```
CHAIN.  
  FIELD: A, B, C.  
  FIELD: D, E, F.  
  MODULE X ON CHAIN-INPUT.  
  MODULE Y.  
ENDCHAIN.
```

Here, module X is called if any of the fields A, B, C, D, E, F has a value different from its initial value. Module Y however is always called. If Y finds an error, *all* six fields are re-opened for input during the error dialog.

To restrict a condition to a particular field, connect the MODULE statement to the relevant FIELD statement.

```
CHAIN.  
  FIELD: A, B, C  MODULE X ON INPUT.  
ENDCHAIN.
```

In this example, module X is called only when the *last* field in the list (C) contains a non-initial value. However, if X finds an error, all three fields (A, B, C) are re-opened for input in the error dialog.

Sometimes you want to make a call depend on several fields in a chain, but not others. For clarity, it is simplest to break up the chain you are using and create separate chains for separate field combinations. In each case, you use ON CHAIN-INPUT or ON CHAIN-REQUEST. For example:

```
CHAIN.  
  FIELD: A, B, C  MODULE X ON CHAIN-REQUEST.  
ENDCHAIN.  
CHAIN.  
  FIELD: A, B, D, E  MODULE Y ON CHAIN-REQUEST.  
ENDCHAIN.
```

Avoiding Automatic Field Checks

Sometimes you want the system to carry out certain processing logic before it performs automatic field checks. For example, if the user wants to exit from the screen, there is no reason to make him or her enter data in the required-input fields.

The flow logic keywords AT EXIT-COMMAND are a special addition to the MODULE statement in the flow logic. AT EXIT-COMMAND lets you call a module before the system executes the automatic field checks:

**** Screen flow logic: ****

```
PROCESS AFTER INPUT.  
  MODULE EXIT AT EXIT-COMMAND.
```

To use AT EXIT-COMMAND, you must assign a function type of E to the relevant function in the Menu Painter or Screen Painter. In the Screen Painter, call up the attributes for the desired pushbutton, and set the attribute *FctType* to E. In the Menu

Issuing Messages

Painter, select *Goto* → *Function list*, and enter *E* in the *Type* column for each function code that should behave as an exit command.

Once you have defined a function as type E, you can use the AT EXIT-COMMAND option to tell the system to process all ABAP/4 modules associated with this function before carrying out any field checks. The AT EXIT-COMMAND event will only be triggered when the user activates a function defined as a type E function.

**** ABAP/4 module: ****

```
MODULE EXIT INPUT.  
  CASE OK_CODE.  
    WHEN 'NEW'.  
      CLEAR: SPFLI, OK_CODE.  
      LEAVE SCREEN.  
    WHEN 'CANC'.  
      CLEAR OK_CODE.  
      SET SCREEN 0. LEAVE SCREEN.  
  ENDCASE.  
ENDMODULE.
```

Normally, the MODULE...AT EXIT-COMMAND statement is intended for processing the exit commands BACK, EXIT and CANCEL. The ABAP/4 module you code to process these commands should contain statements to exit from the screen or transaction, (for example **LEAVE TO SCREEN 0**).

If you do not terminate the screen or transaction in the AT EXIT-COMMAND module, the system continues flow logic processing as usual: first it carries out automatic field checks, then processes the PAI statements in sequential order.

Issuing Messages

An ABAP/4 module lets the system know that an error has occurred by issuing error or warning messages. The message causes the system to enter an error dialog with the user. (The error dialog redisplay the screen and, in the case of an error messages, requires new input.)

Messages are pre-defined texts stored as development objects in the system. The text may contain variable sub-strings you can replace with real texts at runtime. Each message belongs to a message class.

To send messages in a module pool, you must:

- send the message itself (with MESSAGE) and specify the message class in the PROGRAM statement
- if necessary, create the message class and message in the system

The following sections provide information on using messages in ABAP/4:

Sending a Message

Creating a Message Class

Creating Messages

Sending a Message

In general, you send messages using the ABAP/4 statement MESSAGE, and use the message type to signify the type of the error. For example, in the following:

```
IF SY-SUBRC NE 0.  
  MESSAGE E001.  
ENDIF.
```

the message number is 001, and the E is the message type (an error).

You can put five different message types (E, W, I, A, S) on the front of a message number. For example, with message number 001, you could specify:

E001	sends message 001 as a error
W001	sends message 001 as a warning
I001	sends message 001 as an information message
A001	sends message 001 as an abnormal termination message (A=abend)
S001	sends message 001 as a success message

When you put out a message, the error processing that takes place depends on the message type and context. For more information, see *Message Processing in Dialog-Mode*.

Specifying the Message Class

All messages belong to a message class. So if you issue messages in a program, you must also specify a message class. You do this in the PROGRAM statement in your TOP-module:

```
PROGRAM SAPMTZ31 MESSAGE-ID A&.
```

The MESSAGE-ID parameter specifies the two-character code (either alphabetic or numeric) for the message class. This class holds for all messages put out in the module pool. If you need to issue messages from other classes beside the one specified in the PROGRAM statement, give the different class after the message number:

```
MESSAGE E123(ZZ).
```

Remember though that user-defined development objects (such as a message class) should begin with Y or Z.

Adding to the Message Text

Message texts may contain up to four variable texts. When you create the message, you use an & in the text to stand for the variable:

Flight &1 &2 does not exist.

At runtime, the system replaces the strings &1 and &2 with texts provided in the MESSAGE statement. To specify the replacement texts, use the WITH parameter:

```
MESSAGE E005 WITH SPFLI-CARRID SPFLI-CONNID.
```

If the two variables above contain flight number and airline, the user gets the message:

Flight AA 177 does not exist.

Message Processing in Dialog-Mode

In normal dialog processing, a transaction works from screen to screen, processing PBO and PAI events for each.

When a message is issued, screen processing takes a different course, depending on the type of message. We've seen that for an error message, the system redisplay the screen without repeating PBO processing. In the re-display, a message appears in the status line (bottom of screen) and all fields except those causing the error are input-disabled.

In general, the message type determines the screen on which the message appears, and how the dialog proceeds. The differences are:

<i>Message Type</i>	<i>Message appears on</i>	<i>Meaning</i>
Error	Current screen	All screen fields mentioned in FIELD statements are input-enabled. The user must enter a new value. The system then restarts PAI processing for the screen using the new values.
Warning	Current screen	All screen fields mentioned in FIELD statements are input-enabled. However, the user need not enter new values. If new values are entered, the system restarts PAI processing for the screen using the new values. If no new values are entered (the user just presses ENTER), the system resumes PAI processing immediately after the MESSAGE statement
Information	Popup screen	The current screen is interrupted, and the system automatically displays the information message in a popup.

Success	Following screen	The success message is displayed after the PBO processing for the following screen. Success messages thus have no effect on processing of the screen in which they are generated
Abend	Current screen	When the user presses ENTER, the current process is interrupted. The system returns the user to the SAP main menu.

With error messages and warnings, the system restarts PAI processing for the screen after re-displaying it. However, it may not repeat all PAI processing. For information, see *Restarting PAI after an error dialog*.

Restarting PAI after an error dialog

When an ABAP/4 module called with FIELD or CHAIN issues a warning or error message, the system redisplay the screen, gets new input from the user, and then restarts PAI processing. Where this restart takes place (that is, where in the PAI event) may not be immediately obvious. All FIELD or CHAIN statements must be repeated if they share fields with the FIELD or CHAIN statement that found the error. The system determines where to restart using the following steps:

1. In the flow logic, determine which fields:
 - are specified in the CHAIN or FIELD statement that raised the error AND
 - were updated by the user during the last screen display
2. Search through the PAI event for the first FIELD statement that mentions any of the fields in Step 1.
3. Restart processing at that FIELD statement, or if it occurs in a CHAIN, at the beginning of the CHAIN statement.

For example, in the following PAI event:

PROCESS AFTER INPUT.

```
FIELD A MODULE M.
FIELD B MODULE N.
```

```
CHAIN.
  FIELD: A, B, C.
  FIELD: D, E, A.
  MODULE P.
  MODULE Q.
ENDCHAIN.
```

```
CHAIN.
  FIELD: F.
  MODULE R.
ENDCHAIN.
```

```
CHAIN.
```

Issuing Messages

```

FIELD: D.
MODULE S.      ==> ERROR !!!
ENDCHAIN.

```

Module S raises an error. After re-displaying the screen, the system re-starts processing with module P.

Creating a Message Class

You can create a message class from two places in the system:

- from an *Object class* object list (in the Object Browser)
- from an ABAP/4 module (in the ABAP/4 editor)

The most natural method is from the ABAP/4 module containing the PROGRAM statement.

1. Enter the MESSAGE-ID parameter value:

```
PROGRAM SAPMTZ31 MESSAGE-ID A&.
```

2. Double-click on the message-id name (here “A&”).

The system responds with a dialog window asking whether you want to create the message class. Press *Yes*. You jump to the message-class screen.

Display Message Class

Message class Edit Goto Utilities System Help

Message class: A&
 Development class: SDWA
 Last changed by: KEHRERW
 Change date: 03.01.1996 Last changed at: 12:07:17

Attributes

Pflegesprache: D
 Responsible: HILLENBRAND
 Short text: Documentation: Dialog Programming - Examples

3. Enter a *Short text* description and press the *Save* icon.

The message class is created.

4. Press *Messages* to enter actual message texts.

See *Creating Messages* for how to enter message texts.

Creating Messages

There are several ways to jump to a message class to enter message texts. The most natural method is:

1. Code the MESSAGE statement in your program, using the message number you need.

```
MESSAGE E001.
```

If you don't know the next free message number, just enter any number.

2. Double-click on the message number.

The system jumps to the list of messages for the message class. Because message classes can be used by multiple users, it is always better to maintain single messages at a time. Thus, only the message number you requested is in update-mode.

If the message number you entered already exists in the system, you branch to it in display mode. In this case:

- a. Press *Next free number* to jump to the next free message number.
- b. Press *Individual maint.* to edit it.

3. Type in the message text and press the save icon.

Short text	Documentation: Dialog Programming - Examples	
Person responsible	HILLENBRAND	
Last changed by	KEHRERW	
Date	03.01.1996	

Message number		Self-explanator
001	Flight &1 &2 changed	<input type="checkbox"/>
002	Unable to save flight &1 &2	<input type="checkbox"/>
003	Departure airport &1 is unknown	<input type="checkbox"/>
004	Destination airport &1 is unknown	<input type="checkbox"/>
005	Flight &1 &2 not created	<input type="checkbox"/>
006	Flight &1 &2 has no flight data	<input type="checkbox"/>
007	Internal error when reading screen data	<input type="checkbox"/>
008	Unable to display possible entries	<input type="checkbox"/>
009	You are not authorized to display flight data	<input type="checkbox"/>
010	You are not authorized to change flight data	<input type="checkbox"/>

4. Return to your code and correct the message number, if necessary.

If you want to enter several message texts all at once, use the *Maintain all* button to convert all messages to update-mode. Then you don't need to save each message text individually. But remember that you block other users from updating the message class when you do this.

Example Transaction: Checking Field Input

As an example of user-programmed field-checking, we have been using transaction TZ31, one of the example transactions delivered with every system. A quick look at the flow logic and ABAP/4 code for the program demonstrates how the various checks are sewn into the program.

Screen Flow Logic

The screen flow logic (PAI only) for transaction TZ31 looks as follows.

```
*-----*
*   Screen 100: Flow Logic                               *
*&-----*
PROCESS AFTER INPUT.
  MODULE EXIT AT EXIT-COMMAND.
*
  CHAIN.
    FIELD: SPFLI-CARRID, SPFLI-CONNID.
        MODULE CHECK_FLIGHT ON CHAIN-REQUEST.
  ENDCHAIN.
*
  FIELD SPFLI-AIRPFROM
        MODULE CHECK_FR_AIRPORT ON REQUEST.
  FIELD SPFLI-AIRPTO
        MODULE CHECK_TO_AIRPORT ON REQUEST.
*
  MODULE USER_COMMAND_0100.
```

The flow logic is organized so that:

- The statement `MODULE...AT EXIT-COMMAND` handles “exit commands” and occurs at the beginning of PAI. Exit-commands involve processing that must happen *before* the system performs its automatic field checks.
- All `FIELD` and `CHAIN` statements come before any logic that processes user commands. The statements here show that the ABAP/4 modules for checking field values only execute if the `ON REQUEST` condition is fulfilled.

ABAP/4 Code

The following are the TZ31 modules relevant to field-checking. The first, the `EXIT` module, handles the exit-commands, which require no screen input from the user. This includes the `NEW` and `CANC` functions: the `SET SCREEN` and `LEAVE SCREEN` statements cause either the screen or the transaction to terminate. These statements are described in *Controlling the Screen Flow*.

```

*&-----*
*&      Module  EXIT  INPUT
*&-----*
MODULE EXIT INPUT.
  CASE OK_CODE.
    WHEN 'NEW'.
      CLEAR: SPFLI, OK_CODE.
      LEAVE SCREEN.
    WHEN 'CANC'.
      CLEAR OK_CODE.
      SET SCREEN 0. LEAVE SCREEN.
  ENDCASE.
ENDMODULE.

```

The CHECK_FLIGHT module makes sure the flight-number exists for the given airline. If not, the program puts out a message. Note that the SELECT statement alone cannot fill the *Regular* and *Charter* screen fields, since they are represented by a single field in the database. So the module must fill these screen fields explicitly.

```

*&-----*
*&      Module  CHECK_FLIGHT  INPUT
*&-----*
MODULE CHECK_FLIGHT INPUT.
  SELECT SINGLE * FROM SPFLI
    WHERE CARRID = SPFLI-CARRID
    AND CONNID   = SPFLI-CONNID.
  IF SY-SUBRC NE 0.
    MESSAGE E005 WITH SPFLI-CARRID SPFLI-CONNID.
  ENDIF.
  CLEAR: CHARTER, REGULAR.
  IF SPFLI-FLTYPE = SPACE. REGULAR = 'X'.
  ELSE.                CHARTER = 'X'.
  ENDIF.
ENDMODULE

```

Field-checking for the airport fields is as follows.

```

*&-----*
*&      Module  CHECK_AIRPORT  INPUT
*&-----*
MODULE CHECK_FR_AIRPORT INPUT.
  SELECT SINGLE * FROM SAIRPORT
    WHERE ID = SPFLI-AIRPFROM.
  IF SY-SUBRC NE 0. MESSAGE E003 WITH SPFLI-AIRPFROM. ENDIF.
ENDMODULE.

```

After all field-checking has executed, the program can safely use the screen input to process user commands. The module USER_COMMAND_0100 (not shown here) updates the database and puts out a message (either an abort or success), depending on results. Neither of these message types causes the screen processor to enter an error dialog.